## 2. Estado del arte

Actualmente existen diversas alternativas que permiten en mayor o menor medida abstraerse del proveedor de servicios de cloud computing a utilizar. Esto es útil ya que diversos factores pueden hacer que un usuario desee cambiar de proveedor, incluso teniendo su aplicación en producción. Por ejemplo, los costos de un proveedor pueden bajar y convertirlo en una alternativa conveniente. Otro caso podría ser la velocidad de red que posea determinado proveedor para una ubicación geográfica específica. Incluso se podrían usar varios servicios en simultáneo para generar redundancia. Los casos son muchos y todos bien fundados. En las secciones siguientes se presentan brevemente las capacidades de los frameworks existentes con el objetivo de detectar puntos flojos en su diseño. Para este fin, se resaltan las facilidades que posee para cumplir con los siguientes puntos:

* Soporte para las primitivas básicas de storage. Esto significa evaluar si el framework es capaz de subir y bajar archivos, listarlos, extraer sus metadatos o borrarlos
* Capacidad de migrar entre protocolos sin producir mayores cambios en la aplicación. Por ejemplo, una aplicación codificada para S3, puede migrarse a Google Storage? Si es asi, impacta en el código fuente? Puede hacerse de manera simple por archivos de configuración?
* Facilidades para migrar una aplicación ya codificada desde otro framework. Por ejemplo, si en un primer momento la aplicación se codifico contra el SDK de Amazon, se provee algún mecanismo para migrar el código a Google Storage? Si esto no es así el usuario debería recodificar su aplicación desde cero para hacer uso del nuevo servicio.

Se analizaran uno a uno los frameworks actuales, concluyendo el capítulo con una serie de observaciones generales sobre los puntos analizados.

### 2.1. CloudLoop

CloudLoop es un proyecto de código abierto que presenta una API universal y una aplicación de línea de comandos escrita en java para el almacenamiento en cloud. Permite almacenar, administrar y sincronizar los datos entre los proveedores más comunes. Lamentablemente el proyecto se encuentra inactivo e incompleto desde finales del año 2008, lo cual provoca que no sea utilizado en desarrollo de aplicaciones.

La figura 2.1 presenta una vista de alto nivel de la arquitectura de CloudLoop.

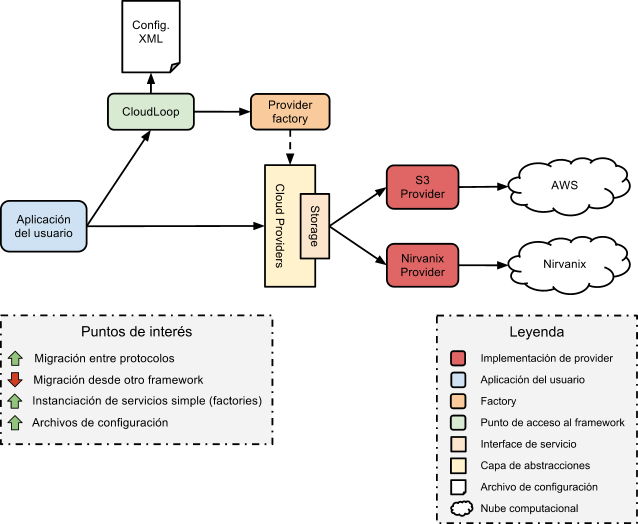


Figura 2.1 Vista de alto nivel de la arquitectura de CloudLoop.

Como se puede apreciar, la configuración de la plataforma se realiza por medio de archivos XML que definen datos básicos del servicio que se desea instanciar (credenciales, directorios, etc). En base a estos archivos de configuración, la fábrica de servicios incluida construye y entrega al usuario una implementación de la interface común de storage. Esto hace que el desarrollador no deba complicarse en la configuración de los objetos particulares de manera manual. Gracias a esto se minimizan los cambios de código en caso de cambios de proveedores.

Las interfaces de CloudLoop soportan todas las primitivas básicas de storage excluyendo la posibilidad de extraer metadatos (solo es posible obtener el tamaño de un archivo). El uso de cada método es sencillo y tiene javadocs claros que acompañan al desarrollador. Como consecuencia de esto, la implementación de un nuevo proveedor es sencilla y tan solo se debe implementar una interface clara que define todos los métodos comunes a los diversos proveedores de storage.

La plataforma también permite cambiar el proveedor de servicios de forma muy simple ya que casi no es necesario realizar modificaciones en el código fuente de la aplicación. Para esto tan solo se debe cambiar la configuración por medio del XML de configuración, definiendo todos los datos necesarios para el nuevo servicio a utilizar.

Uno de los puntos flojos de la plataforma es la imposibilidad de migrar desde otro framework. En estos casos el desarrollador no tiene otra salida que recodificar la aplicación para adaptarse al modelo de CloudLoop.

**Servicios de storage soportados**

* Amazon S3
* Nirvanix

**Ventajas y desventajas**

**-** Proyecto inactivo desde el año 2008.

**+** Soporte para storage en Java.

**-** No soporta todas las primitivas básicas de storage. Es imposible extraer metadatos de un archivo (salvo su tamaño).

**+** El uso de las primitivas es simple y claro.

**+** Es fácil migrar entre protocolos dentro del framework.

**+** Es sencillo implementar un nuevo servicio dentro de CloudLoop.

**-** No es posible migrar desde otro framework sin recodificar toda la aplicación.

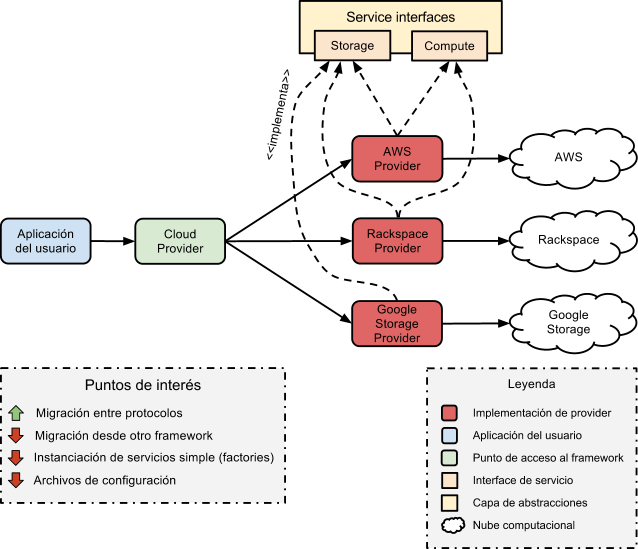
**+** Posee XMLs de configuracion.

**+** La instanciación de los servicios se facilita por medio del uso de factories.

### 2.2. Dasein

Dasein es un proyecto introducido en 2009 por la empresa enStratus Networks LLC. Provee abstracciones para servicios de los tipos compute y storage, aunque existen otras implementaciones para servicios menores de diversos proveedores. El proyecto se encuentra activo.

La figura 2.2 presenta una vista de alto nivel de la arquitectura de Dasein.



[Figura 2.2](http://dasein-cloud.sourceforge.net/) Vista de alto nivel de la arquitectura de Dasein.

Podemos notar que en la arquitectura de Dasein no se contempla el uso de archivos de configuración o mecanismos de construcción de instancias de servicios. La configuración de los servicios y nubes se realiza completamente por código. En la misma sintonía, el desarrollador que utilice Dasein deberá codificar la instanciación de los servicios deseados, teniendo que involucrarse con las particularidades de cada servicio.

La plataforma soporta la mayoría de las primitivas básicas para storage. Pese a esto, Dasein no provee mecanismos para copiar o buscar archivos mediante un sistema de prefijos o expresiones regulares. Utilizar y descubrir los métodos a utilizar no es simple dada la falta total de documentación, javadocs o comentarios de código. El usuario debe imaginarse que clase hace que cosa y de que manera lo hace, todo a prueba y error. Es habitual cruzarse con parámetros poco claros. Por ejemplo, en el método “upload” que se utiliza para subir un archivo, se encuentra el parámetro “multipart”. Este parámetro se encuentra en una interface pero solo tiene sentido para Amazon S3 (habilita multipart upload[[1]](#footnote-1)). Pese a todo esto, una vez que se comprende que se debe utilizar y como, el framework trabaja correctamente.

En la misma línea, la implementación de un nuevo proveedor tiene algunas complicaciones. Se deben generar implementaciones para la nube (ej: Amazon) y el servicio particular (ej: S3). Si bien esto no supone gran complejidad, descubrir que debe hacer cada método que se debe implementar no es trivial. Otra vez se vuelve a caer en la falta de documentación de la plataforma.

En cuanto a la migración entre proveedores dentro de Dasein podemos notar que es posible sin hacer demasiadas modificaciones al código. Para esto es necesario cambiar la nube que se está utilizando y reconfigurar sus propiedades manualmente.

Si hablamos de migrar hacia Dasein desde un producto alternativo el resultado no es positivo. Dasein no provee ningun mecanismo para facilitar este tipo de migración y el usuario se ve obligado a recodificar su aplicación utilizando la nueva tecnología.

**Servicios de storage soportados**

* Amazon S3
* GoGrid
* Rackspace
* ReliaCloud
* Azure
* AT&T Synaptic Storage

**Ventajas y desventajas**

**+** Desarrollo activo al año 2011.

**+** Soporte para storage en Java.

**-** No soporta todas las primitivas básicas de storage. No se posibilita copiar y buscar archivos.

**-** El uso de las primitivas es complejo por falta de documentación.

**+** Es fácil migrar entre protocolos dentro del framework.

**-** Existen complicaciones para implementar un nuevo servicio dentro de libcloud.

**-** No es posible migrar desde otro framework sin recodificar toda la aplicación.

**-** No posee archivos de configuración, todo se realiza por código.

**-** No se poseen factories, el usuario debe construir todos los servicios a mano.

### 2.3. Deltacloud

Deltacloud es un proyecto en incubación de la Fundación Apache. Se distingue del resto de los proyectos de su tipo gracias a su arquitectura cliente-servidor basada en una API REST. Gracias a esto se puede hacer uso de Deltacloud desde cualquier lenguaje de programación. Los desarrolladores mantienen un cliente CLI para Ruby y existen bases de un cliente Java, aunque por el momento solo soporta servicios de tipo compute. Si no es posible utilizar Ruby o si la funcionalidad en java es insuficiente, se debe generar un nuevo cliente implementando la API de Deltacloud. Esto se traduce en un incremento en el tiempo y costo de desarrollo. El proyecto se encuentra activo.

La figura 2.3 presenta una vista de alto nivel de la arquitectura de Deltacloud.

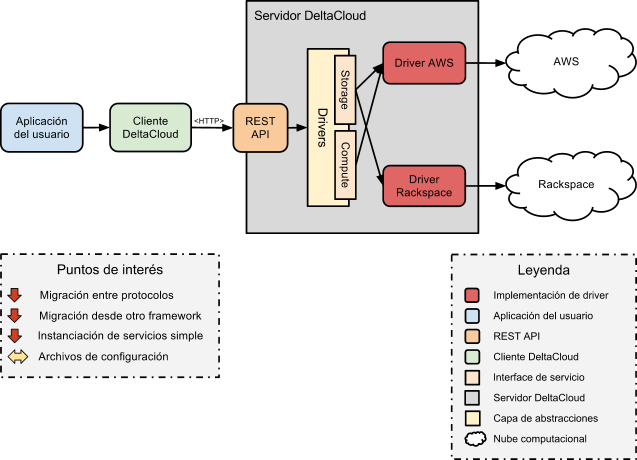


Figura 2.3 Vista de alto nivel de la arquitectura de DeltaCloud.

Como se puede apreciar, Deltacloud no posee por defecto archivos de configuración o mecanismos de instanciación de servicios. La configuración de los servicios se realiza en el cliente y esta librada a la implementación particular de la API, que suelen contar con estos mecanismos. El servidor es stateless por lo que no almacena credenciales u otros datos que deben ser provistos en cada invocación. Para instanciar un servicio se debe configurar tanto el servidor como el cliente. El servidor para que levante el servicio puntual y el cliente para enviar los datos del nuevo servicio (credenciales, etc).

En cuanto al soporte de las funciones básicas de storage, Delta Cloud tiene algunas limitaciones. Solo se provee soporte para bajar, subir y eliminar archivos. Por el momento no es posible listar, mover o copiar archivos. Las primitivas están definidas en una API clara y con buena documentación. Esto último facilita la tarea a la hora de implementar un nuevo servicio. Para lograr esto el desarrollador debe crear un nuevo Driver definiendo el tipo de servicio que se está implementando (storage, compute) lo que define el conjunto de funciones a implementar.

La migración entre protocolos puede ser un problema para la plataforma ya que no es posible levantar un servidor multi cloud. Esto genera que sea necesario contar con varios servidores en varios puertos si se desea trabajar con más de una nube. Esto provoca que el desarrollador tenga que modificar tanto el cliente (envio de credenciales, reconfiguracion de parametros de conexion, etc) como el servidor (reconfiguracion) en el caso de querer migrar de un proveedor a otro.

Delta Cloud tampoco se provee ningún tipo de mecanismo para facilitar la migración desde un producto competidor (por ejemplo, desde jClouds) por lo que el desarrollador debe recodificar su aplicación.

**Servicios de storage soportados**

* Amazon S3
* Rackspace
* Azure
* Walrus

**Ventajas y desventajas**

**+** Proyecto activo.

**-** No se soporta storage en Java.

**-** No soporta todas las primitivas básicas de storage. Solo soporta bajar, subir y borrar archivos.

**+** El uso de las primitivas está bien definido.

**-** La migración puede resultar confusa, afectando tanto cliente como servidor.

**+** Es sencillo implementar un nuevo servicio gracias a una documentación precisa.

**-** No es posible migrar desde otro framework sin recodificar toda la aplicación.

**+/-** La configuración del cliente depende de la implementación particular de la API. El servidor (CLI) debe levantarse nuevamente con la nueva configuración para reflejar los cambios.

**-** La instanciación de un servicio impacta tanto cliente como servidor.

### 2.4. Java-Storage

Este proyecto tiene como objetivo permitir el acceso a diferentes servicios de storage de forma genérica. Se distingue del resto de los proyectos por proveer una re implementación de “File” como interface principal. El proyecto se encuentra congelado desde finales de 2009.

La figura 2.4 presenta una vista de alto nivel de la arquitectura de Java Storage.

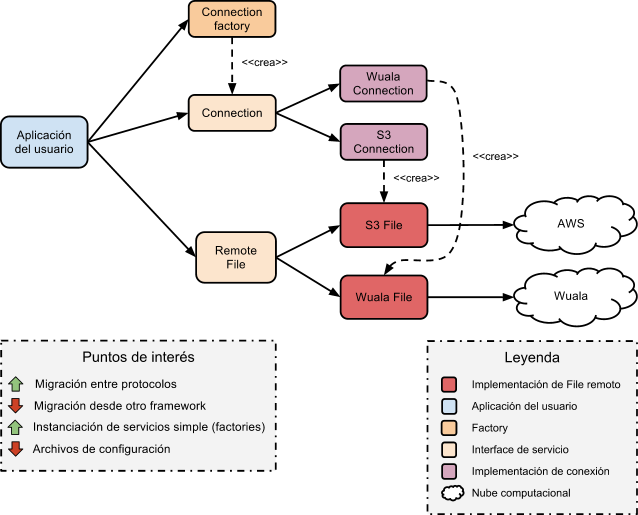


Figura 2.4 Vista de alto nivel de la arquitectura de Java-Storage.

Como se puede apreciar, Java-Storage no cuenta con archivos de configuración para los servicios que soporta. La configuración de los servicios se realiza completamente por código, por lo que cualquier cambio impacta en el código del usuario. Pese a esto, se debe notar la existencia de una fábrica que facilita la instanciación de los File remotos y sus conexiones. Esta fábrica retorna instancias de objetos similares a File por lo que el desarrollador seguramente ya sepa qué hacer con ellas.

Java-Storage soporta métodos básicos para descargar y listar archivos, pero carece totalmente de soporte para subir archivos. Esto se suma a la falta de soporte para metadatos, listar, borrar, mover o copiar archivos. Si hablamos de implementar un nuevo protocolo tan solo es necesario implementar una serie de interfaces bien definidas. A esto contribuye que el usuario de Java típico ya está familiarizado con el uso de File y entiende que debería hacer cada método.

La migración entre protocolos dentro del framework es simple. Tan solo debe cambiarse la instancia de File por la requerida para el nuevo protocolo.

Otro de los puntos flojos del proyecto es que tampoco cuenta con facilidades para migrar una aplicación desde otro framework, generando que el desarrollador deba codificar nuevamente su aplicación.

**Servicios de storage soportados**

* Wuala
* Amazon S3

**Ventajas y desventajas**

**-** Proyecto inactivo desde el año 2009.

**+** Soporte para storage en Java.

**-** No soporta todas las primitivas básicas de storage. Tan solo soporta download.

**+** El uso de las primitivas es simple y claro por utilizar File de base.

**+** Es fácil migrar entre protocolos dentro del framework.

**+** Es simple implementar un nuevo servicio para Java-Storage.

**-** No es posible migrar desde otro framework sin recodificar toda la aplicación.

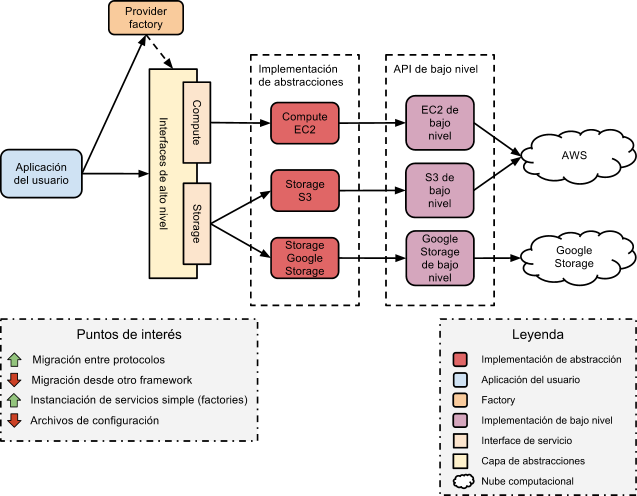
**-** No posee archivos de configuración.

**+** Se utilizan factories para instanciar los Files remotos.

### 2.5. jClouds

jClouds es un framework de código abierto muy útil para comenzar a trabajar en la nube y provee abstracciones para los servicios de compute y storage. En su desarrollo se tuvo en cuenta la performance y el consumo de recursos ya que fue diseñada también para ser utilizada en ambientes limitados como Google Apps. El proyecto se encuentra activo y es muy usado, incluso por otros frameworks que utilizan sus implementaciones de servicios de bajo nivel.

La figura 2.5 presenta una vista de alto nivel de la arquitectura de jClouds.



[Figura 2.5](http://code.google.com/p/jclouds/) Vista de alto nivel de la arquitectura de jClouds.

jClouds posee una arquitectura simple, pero bien llevada a cabo. La configuración de los servicios se realiza por medio de código, pero el uso de factories con interfaces simples para realizar las instanciaciones hace que la configuración se limite a credenciales que pueden fácilmente volcarse en un archivo properties.

La plataforma soporta la totalidad de las primitivas de storage por medio de interfaces simples y bien documentadas. En el caso de querer acceder a funcionalidad de bajo nivel jClouds brinda la posibilidad de acceder a la API subyacente con todos sus métodos particulares. Siguiendo la misma tendencia, jClouds hace que la tarea de implementar un nuevo protocolo sea clara y sin mayores complicaciones. Solo se debe implementar la interface definida para storage. Los desarrolladores de jClouds mantienen documentación simple pero suficiente para esta y otras tareas relacionadas al desarrollo del framework.

Migrar entre protocolos dentro de jClouds es muy simple. Esto es así gracias al uso de factories que hacen que cambiar de protocolo resulte tan complejo como cambiar el set de credenciales y demás elementos de configuración del nuevo protocolo.

Uno de los puntos que jClouds no considera es la posibilidad de migrar desde un framework alternativo. El desarrollador se ve obligado a re implementar su aplicación para acceder a las bondades de esta plataforma.

**Servicios de storage soportados**

* Atmos
* Azure
* Rackspace
* S3
* CloudFiles
* Synaptic
* Scaleup
* Google Storage
* Ninefold
* Tiscali

**Ventajas y desventajas**

**+** Proyecto activo.

**+** Soporte para storage en Java.

**+** Soporta todas las primitivas básicas de storage.

**+** El uso de las primitivas es simple y claro.

**+** Es fácil migrar entre protocolos dentro del framework.

**+** La manera de implementar un nuevo servicio es simple y clara.

**-** No es posible migrar desde otro framework sin recodificar toda la aplicación.

**-/+** No posee archivos de configuración, pero estos no tienen mayor impacto gracias al buen uso de factories para instanciar los servicios.

**+** La instanciación de los servicios se facilita por medio del uso de factories.

### 2.6. JetS3t

JetS3t es un conjunto open-source de herramientas y aplicaciones para los servicios de Amazon Simple Storage Service (Amazon S3) y Google Storage. Proporciona a los programadores Java una API potente y sencilla para interactuar con los servicios de almacenamiento y gestión de datos almacenados. JetS3t se encuentra activo y cuenta con una gran base de usuarios (principalmente como conector a S3).

La figura 2.6 presenta una vista de alto nivel de la arquitectura de JetS3t.

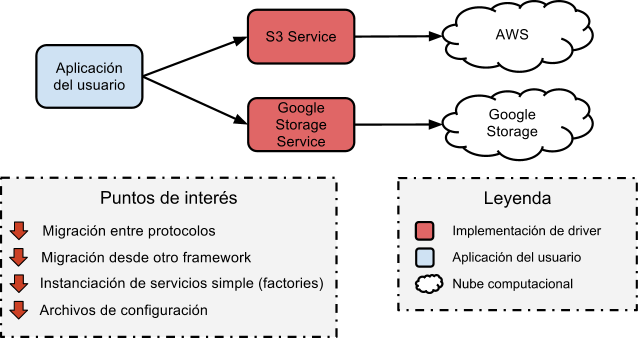


Figura 2.6 Vista de alto nivel de la arquitectura de JetS3t.

Como se puede apreciar, la arquitectura de JetS3t es muy simple. No se cuentan con mecanismos para instanciar objetos fácilmente, ni facilidades para configurar los servicios por medio de archivos o mecanismos similares. Esto provoca que cualquier cambio de proveedor o de configuración de los mismos impacte sobre el código del usuario.

La plataforma soporta todas las primitivas básicas de storage. Su uso es sencillo, en particular para desarrolladores con conocimientos de la API de Amazon S3, en la cual JetS3t basa su modelo. De esto último surge el inconveniente más grande de JetS3t y es que no fue concebido como framework. El agregado de Google Storage generó un set aislado de objetos y diversos métodos que cambian su comportamiento utilizando “instanceof” para reconocer S3 de Google Storage. Esto provoca que implementar un nuevo protocolo no sea una tarea trivial y que la migración entre proveedores tenga dificultades y no sea un simple cambio de credenciales.

El proyecto tampoco cuenta con facilidades para migrar una aplicación desde otro framework, generando que el desarrollador deba codificar nuevamente su aplicación.

**Servicios de storage soportados**

* Amazon S3
* Google Storage

**Ventajas y desventajas**

**+** Proyecto activo y con buena base de usuarios.

**+** Soporte para storage en Java.

**+** Soporta todas las primitivas básicas de storage.

**+** El uso de las primitivas es simple y claro.

**-** Presenta complicaciones para migrar entre protocolos dentro del framework.

**-** No es simple implementar un nuevo protocolo dentro de JetS3t.

**-** No es posible migrar desde otro framework sin recodificar toda la aplicación.

**-** La configuración se realiza totalmente por código.

**-** No se poseen factories, el usuario debe construir todos los servicios a mano.

### 2.7. libcloud

El proyecto libcloud tiene por objetivo la generación de una interface única y simple a los servicios de tipo “compute” de distintos proveedores. Además de esto también cuenta con soporte sencillo para servicios de storage. Es principalmente un módulo para Python, pero también cuenta con desarrollo para Java, aunque por el momento con capacidades reducidas. El desarrollo de la versión Java de libcloud se encuentra congelado desde noviembre de 2010.

La figura 2.7 presenta una vista de alto nivel de la arquitectura de libcloud.

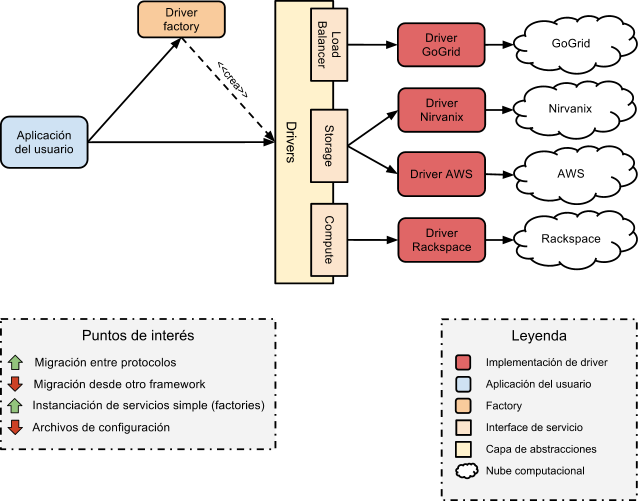


Figura 2.7 Vista de alto nivel de la arquitectura de libcloud.

Como se puede apreciar libcloud posee una fábrica de drivers que gestiona la instanciación de los objetos servicio deseado. Esto evita que el desarrollador tenga que conocer detalles de bajo nivel de la construcción de cada objeto. Sin embargo, la configuración de cada servicio se realiza por código sin ayuda de archivos XML o properties. Esto genera que cualquier cambio en los parámetros de un servicio traigan aparejado cambios de código.

En cuanto a su funcionalidad, libcloud soporta todas las primitivas básicas (subir, bajar, listar, buscar, borrar y extraer metadatos). Para comenzar a trabajar con libcloud para storage tan solo tenemos que crear instancias de alguna implementación de la interface de storage. Los métodos provistos presentan un set suficiente y claro de primitivas para trabajar con cada proveedor. Es igualmente simple implementar un nuevo servicio de storage. Para esto debe extenderse la interface provista implementando todos los métodos necesarios.

Si hablamos de migración entre proveedores dentro de libcloud, podemos notar que si se utilizan correctamente las interfaces provistas el pasaje es muy simple. Tan solo se deben cambiar el tipo de la clase concreta y los parámetros necesarios del constructor.

Un punto que libcloud no contempla es su utilización dentro de una aplicación ya codificada con otro framework. Por ejemplo, si el usuario tiene implementada una aplicación para S3 con JetS3 y quiere utilizar Nirvanix con libcloud no tiene otra salida que volver a codificar su aplicación desde cero.

**Servicios de storage soportados**

* Nirvanix
* S3

**Ventajas y desventajas**

**-** Desarrollo congelado desde fines de 2010.

**+** Soporte para storage en Java.

**+** Soporta todas las primitivas básicas de storage.

**+** El uso de las primitivas es simple y claro.

**+** Es fácil migrar entre protocolos dentro del framework.

**+** Es sencillo implementar un nuevo servicio dentro de libcloud.

**-** No es posible migrar desde otro framework sin recodificar toda la aplicación.

**-** No posee archivos de configuración, todo se realiza por código.

**+** Hace uso de factories para facilitar la instanciación de servicios.

### 2.8. Resultados generales

La Tabla 2.1 presenta los detalles básicos de cada aplicación. Es interesante resaltar que tres de los proyectos evaluados no se encuentran en actividad (libcloud, CloudLoop y Java-Storage).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Proyecto** | **Activo?** | **Soporte para Java?** | **Documentación** | **URL** |
|
| **libcloud** | No (Java) | Si | Buena | [http://libcloud.apache.org](http://libcloud.apache.org/) |
| **jClouds** | Si | Si | Buena | <http://code.google.com/p/jclouds/> |
| **Dasein** | Si | Si | Muy pobre | <http://dasein-cloud.sourceforge.net/> |
| **Deltacloud** | Si | Parcial, solo para Compute | Buena | <http://incubator.apache.org/deltacloud/index.html> |
| **JetS3t** | Si | Si | Buena | <https://github.com/geemus/fog> |
| **CloudLoop** | No | Si | Regular | <https://cloudloop.dev.java.net/> |
| **Java-Storage** | No | Si | Buena | <http://www.simplecloud.org/api> |

Tabla 2.1 Información básica

Datos de mayor interés aparecen en la tabla 2.2. Como se puede apreciar, la mayoría de los productos evaluados soportan todas las primitivas básicas de storage. Es notable que Java-Storage solo soporte descargar archivos y que DeltaCloud solo permita subir, descargar o borrar. Podemos ver que en general el uso de las primitivas es sencillo, a excepción de Dasein que dada la inconsistencia de sus interfaces y su escasa documentación puede tornarse problemático.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Proyecto** | **Servicios de Storage soportados** | **Soporte de primitivas basicas** | | | | | | **Facilidad en el uso de las primitivas** |
| **Subir** | **Bajar** | **Listar** | **Buscar** | **Borrar** | **Metadatos** |
| **libcloud** | Nirvanix, S3 | Si | Si | Si | Si | Si | Si | Facil |
| **jClouds** | Atmos, Azure, Rackspace, S3, CloudFiles, Google Storage, etc | Si | Si | Si | Si | Si | Si | Facil |
| **Dasein** | S3, GoGrid, Rackspace, ReliaCloud, Azure, AT&T | Si | Si | Si | No | Si | Si | Dificil |
| **Deltacloud** | S3, Rackspace, Azure, Walrus | Si | Si | No | No | Si | No | Facil |
| **JetS3t** | S3, Google Storage | Si | Si | Si | Si | Si | Si | Facil |
| **CloudLoop** | Nirvanix, S3 | Si | Si | Si | Si | Si | No | Facil |
| **Java-Storage** | S3, Wuala | No | Si | No | No | No | No | Facil |

Tabla 2.2 Soporte para servicios de tipo storage storage

Los datos más relevantes se presentan en la tabla 2.3. Es destacable que ninguna de las herramientas posee facilidades para migrar desde otro framework. Esto obliga a que el desarrollador deba recodificar su aplicación en caso de querer migrar. Otro de los datos interesantes es que salvo CloudLoop, ninguna de las herramientas trabaja la instanciación y configuración de servicios por medio de archivos y factories o mecanismos similares. En estos casos, el código debe modificarse cuando se desea cambiar de proveedor, credenciales, etc. Tambien se puede apreciar que en general se puede decir que es simple implementar nuevos servicios en todos los frameworks sacando Dasein y JetS3t. Estos últimos productos tienen deficiencias en la definición de sus interfaces y/o documentación que hacen que implementar un nuevo servicio no sea una tarea trivial.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Proyecto** | **Facilidad para implementar servicios** | **Facilidad para migrar entre protocolos** | **Provee mecanismos para migrar desde otro framework** | **Provee archivos de configuración** | **Provee factories para instanciar servicios** |
| **libcloud** | Facil | Facil | No | No | No |
| **jClouds** | Facil | Facil | No | No | No |
| **Dasein** | Dificil | Facil | No | No | No |
| **Deltacloud** | Facil | Dificil | No | No | No |
| **JetS3t** | Dificil | Dificil | No | No | No |
| **CloudLoop** | Facil | Facil | No | Si | Si |
| **Java-Storage** | Facil | Facil | No | No | No |

Tabla 2.1 Capacidades de migración, configuración e inclusión de nuevos servicios

1. Metodo de upload utilizado por Amazon para soportar archivos mayores a 5GB, dividiendo el archivo en pequeños pedazos que se suben y almacenan por separado. <http://aws.typepad.com/aws/2010/11/amazon-s3-multipart-upload.html> [↑](#footnote-ref-1)